

# Cleartext Password Elimination

David E. Tweten

September 22, 2000

### **Abstract**

With the advent of sniffers and sniffer kits, it is no longer acceptable for durable passwords to be used in cleartext form for user authentication across any network, even the NAS internal LAN. This plan details a way to banish cleartext passwords from the NAS net for the kind of authentication that is most vulnerable to sniffers.

# Contents

|   |            |
|---|------------|
| <b>Acknowledgments</b>  | <b>iii</b> |
| <b>1 Introduction</b>   | <b>1</b>   |
| 1.1 Background and Motivation . . . . .                         | 1          |
| 1.1.1 Types of Authentication . . . . .                         | 2          |
| 1.2 The Problem, Real and Imagined . . . . .                    | 2          |
| 1.2.1 The Imagined Problem: Hostile Keyboard Monitors . . . . . | 2          |
| 1.2.2 The Real Problem: Hostile Sniffers . . . . .              | 3          |
| <b>2 Solutions</b>  | <b>5</b>   |
| 2.1 The Hardware Solutions . . . . .                            | 5          |
| 2.2 The Software Solutions . . . . .                            | 6          |
| 2.2.1 Kerberos . . . . .  | 6          |
| 2.2.2 S/Key . . . . .   | 6          |
| 2.2.3 Secure Shell (Ssh) . . . . .                              | 7          |
| 2.2.4 The Secure Remote Password Protocol . . . . .             | 8          |
| 2.2.5 Other Software Solutions . . . . .                        | 8          |
| 2.3 The Real Solution - A Software Mix . . . . .                | 10         |
| <b>3 The Preferred Configuration</b>                            | <b>11</b>  |
| 3.1 Unix Ssh Do and Don't List . . . . .                        | 11         |
| 3.1.1 Place Ssh Under /usr/prg, Not Under /usr/local . . . . .  | 11         |
| 3.1.2 Enable X Forwarding . . . . .                             | 11         |
| 3.1.3 Disable the IDEA Cipher . . . . .                         | 12         |
| 3.1.4 Enable DES . . . . .                                      | 12         |
| 3.1.5 Enable 3DES . . . . .                                     | 12         |
| 3.1.6 Build Ssh With libwrap . . . . .                          | 12         |
| 3.1.7 Build Ssh Without RSAREF on Government Machines . . . . . | 13         |
| 3.1.8 Make Blowfish the Default Cipher . . . . .                | 13         |
| 3.2 OPIE . . . . .  | 13         |
| 3.2.1 OPIE Calculators . . . . .                                | 14         |
| 3.3 FTP . . . . .   | 14         |
| 3.3.1 Standard NAS FTP Configuration . . . . .                  | 14         |
| 3.3.2 Bastion Host FTP Configuration . . . . .                  | 15         |

|          |   |           |
|----------|---|-----------|
| 3.4      | Macintosh Configuration . . . . .                     | 15        |
| 3.5      | Windows 95/98/NT/2K Configuration . . . . .           | 16        |
| <b>4</b> | <b>The Usage Model</b>                                | <b>19</b> |
| 4.1      | General Issues . . . . .                              | 19        |
| 4.2      | Trusted Unix Machines . . . . .                       | 21        |
| 4.3      | MacOS . . . . .                                       | 24        |
| 4.4      | Windows 95/98/NT/2K . . . . .                         | 25        |
| 4.5      | Untrusted Machines . . . . .                          | 27        |
| <b>5</b> | <b>The Tasks to be Performed</b>                      | <b>29</b> |
| 5.1      | Install Ssh Protocol 1.5 Everywhere . . . . .         | 29        |
| 5.1.1    | Unix Configuration . . . . .                          | 34        |
| 5.1.2    | Personal Computer Configuration . . . . .             | 35        |
| 5.2      | Collect Public Machine Keys Automatically . . . . .   | 35        |
| 5.3      | Check Users' Ssh Machine List . . . . .               | 36        |
| 5.4      | Distribute Helper Scripts . . . . .                   | 36        |
| 5.5      | Install Modified OPIE on Bastions . . . . .           | 37        |
| 5.6      | Educate Users . . . . .                               | 37        |
| 5.7      | Make Administration Utilities Work With Ssh . . . . . | 38        |
| 5.7.1    | PCP Modification for Ssh . . . . .                    | 38        |
| 5.7.2    | PBS Configuration for Ssh . . . . .                   | 38        |
| 5.8      | Disable Cleartext Passwords . . . . .                 | 38        |
| 5.9      | Schedule . . . . .                                    | 39        |
| <b>A</b> | <b>ISO 9001 Considerations</b>                        | <b>41</b> |
| A.1      | Software Project Characterization . . . . .           | 41        |
| A.1.1    | Software Project Risk Classification . . . . .        | 41        |
| A.1.2    | Software Project Cost Classification . . . . .        | 41        |
| A.1.3    | Organizational Complexity Classification . . . . .    | 42        |
| A.2      | Selected Development Controls . . . . .               | 42        |
| A.2.1    | Development Approach . . . . .                        | 42        |
| A.2.2    | Documentation Plan . . . . .                          | 42        |
| A.2.3    | Maintenance Plan . . . . .                            | 42        |
| A.2.4    | Software Configuration Control . . . . .              | 43        |
| A.2.5    | Change Request Tracking . . . . .                     | 43        |
| A.2.6    | Required Reviews . . . . .                            | 43        |
| A.2.7    | Replication, Delivery, and Installation . . . . .     | 43        |
|          | <b>Bibliography</b>                                   | <b>44</b> |

# Acknowledgments

In the lead-up to its ISO Software Project Plan Review meeting, this plan was submitted to a variety of people for comment, both individually and by e-mail list. Many responded with quite valuable suggestions that materially improved this document's quality. The individuals and e-mail lists appear in Table 1, people first, in alphabetical order, and with asterisks after the names of those who provided comments. Thank you, all.

| <i>Name</i>          | <i>E-Mail</i>                    |
|----------------------|----------------------------------|
| PAUL ALLEN*          | paul.l.allen@boeing.com          |
| LOUIS J. BLAZY       | Louis.J.Blazy@ivv.nasa.gov       |
| DAVE BOYLE*          | dave.boyle@ae.ge.com             |
| RANDY BUTLER*        | rbutler@ncsa.uiuc.edu            |
| LAURA CARRIERE*      | laura@dao.gsfc.nasa.gov          |
| CHRISTINE CORTEZ*    | ccortez@nas.nasa.gov             |
| ROGER DANIELS*       | roger.daniels@ae.ge.com          |
| JAMES R. FISCHER     | jfischer@pop900.gsfc.nasa.gov    |
| KEN FREEMAN          | kfreeman@mail.arc.nasa.gov       |
| DOUGLAS M. FRIEDMAN* | Douglas.Friedman@West.Boeing.com |
| BRIAN J. GLASS       | bglass@arc.nasa.gov              |
| KEN HORNSTEIN*       | kenh@cmf.nrl.navy.mil            |
| MATTHEW JACOB*       | mjacob@nas.nasa.gov              |
| RICHARD JAFFE*       | jaffe@pegasus.arc.nasa.gov       |
| DENNIS C. JESPERSEN* | jesperse@nas.nasa.gov            |
| BOB MOHLENHOFF*      | bmohlenhoff@mail.arc.nasa.gov    |
| NANCY L. PALM        | Nancy.L.Palm.1@gsfc.nasa.gov     |
| TOM PERRINE*         | tep@sdsc.edu                     |
| HELMUTH PESCADOR     | hpescador@mail.arc.nasa.gov      |
| DAVID PICASSO        | dpicasso@mail.arc.nasa.gov       |
| HARPER PRYOR         | harper@dao.gsfc.nasa.gov         |
| JOHN RAY             | jrray@mail.arc.nasa.gov          |
| DAVE RUDY*           | d.h.rudy@larc.nasa.gov           |
| GEORGE R. RUMNEY II* | rumney@amarna.gsfc.nasa.gov      |
| CATHY SCHULBACH      | cschulbach@mail.arc.nasa.gov     |
| MIKE STONE*          |                                  |
| GEOFF M. TENNILLE*   | g.m.tennille@larc.nasa.gov       |
| LINDA THOMPSON*      | lthompson@jean-luc.arc.nasa.gov  |
| EUGENE TU            | eltu@mail.arc.nasa.gov           |
| STEVE TUECKE*        | tuecke@mcs.anl.gov               |
| PAM WALATKA*         | walatka@nas.nasa.gov             |
| ALEX WOO*            | awoo@mail.arc.nasa.gov           |
|                      | hec-users-all@nas.nasa.gov       |
|                      | ssh-admin@nas.nasa.gov           |
|                      | ssh-users@nas.nasa.gov           |
|                      | user-group@nas.nasa.gov          |

Table 1: Pre-approval reviewers

# Chapter 1

## Introduction

This initial chapter covers some background, defines the problem to be solved and gives a quick overview of the structure of the rest of this report so you can decide which parts you need to read.

Chapter 2 tells why we chose the solutions we did and why we didn't choose the others. Chapter 3 defines in great detail how we intend to configure and use the solution we chose. Chapter 4 describes the usage experience for Unix, MacOS, and Windows users and for users accessing NAS resources from an insecure machine. Chapter 5 describes the tasks that must be performed to complete the project. Finally, Appendix A fulfills the requirements of NASA Ames ISO 9001 procedures.

### 1.1 Background and Motivation

Authentication is the process of proving you are who you claim to be. You can accomplish that by proving that you know something, by proving that you have something, or by proving that you are something. In each case, "something" is a thing previously known to the authenticating entity. Password authentication, cryptographic authentication, and zero-knowledge proofs all demonstrate that you know something. Keys, whether mechanical or electronic, demonstrate that you have something. Biometric authentication systems allow you to demonstrate that you are something. Unix tradition is to demonstrate that you know something, typically a password.

Passwords, `/etc/hosts.equiv`, and `~/.rhosts` files have been used for Unix user authentication since the dawn of Unix time<sup>1</sup>, or the beginning of Unix networking, as appropriate. They have accomplished two kinds of authentication, one of which has recently become vulnerable to hostile sniffers. For the vulnerable kind of authentication, we must now move on from password authentication to something(s) less vulnerable.

---

<sup>1</sup>00:00:00 GMT, January 1, 1970.

### 1.1.1 Types of Authentication

For purposes of this project, there are two main types of user authentication, local and remote. For local authentication, a user presents himself physically to a user-trusted machine, proposing to use its directly-connected input or output devices to communicate with programs running on that machine. Remote authentication takes place when a program on the user-trusted machine, running on behalf of the user, tries to convince another, network-connected machine that it should grant the user access to its programs. There is a third situation of interest, when there is no user-trusted machine available, but the user must still authenticate himself to a distant machine. Methods of authentication appropriate to each situation are not necessarily appropriate to the others, yet the Unix community has thus far been using cleartext passwords for all.

For purposes of local authentication, a user can present himself to a machine's console or to any terminal connected to the machine. Any of the three basic proofs can be used for authentication.

Remote authentication is only possible by demonstrating that the user's machine "knows" something that should only be known to the user. The user's machine can't demonstrate that the user has or is something because neither the user nor his possessions are available to the remote authenticating machine for examination.

If the user can't trust the local machine being used to accomplish remote authentication, then some sort of zero-knowledge proof must be used. Otherwise, it would be possible for a hostile local machine to harvest the user's knowledge so someone else could later accomplish a fraudulent authentication.

In the current risk environment, traditional Unix means of authentication are still sufficient for local authentication. They are no longer sufficient for remote authentication, whether done through the services of a user-trusted local machine, or not.

## 1.2 The Problem, Real and Imagined

To attack a password authentication system, you need to harvest users' passwords. In principle, it ought to be possible to do so at any point after the users' fingers strike a keyboard and until the password checking program destroys its input. In fact, the risk can be made acceptably small within a properly configured machine. The risk is not nearly so controllable once a cleartext password is sent over a network.

### 1.2.1 The Imagined Problem: Hostile Keyboard Monitors

It is certainly possible to install keyboard monitor software on a machine. A keyboard monitor is a program that reads anything typed by the user in parallel with whatever program the user intended to receive his input. It is even possible to "install" one remotely on poorly configured Unix X machines.

Any privileged (root) program on a Unix machine can read kernel memory. Users' keystrokes are stored temporarily in kernel memory until the characters can be delivered to the intended application. All a cracker has to do is gain the ability to run as root on a target machine, and he can set up a keyboard monitor. On many machines, it is sufficient to be able to run as a member of the group, kmem. If anything, this is easier than setting up a sniffer.

It isn't even necessary to be able to run as the root user or as a member of the kmem group on many machines that run X. Many such machines are configured to use so called "host" authentication rather than the more secure "xauth" authentication. The X server on the user's workstation "serves" access to the user's keyboard, mouse, and screen. Any X application running on a host not excluded under "host" authentication or any application able to masquerade as having come from such a host can read the X server's keyboard without the user's knowledge. The obvious countermeasure is to use "xauth" authentication on all X servers.

Though possible and easy to install, keyboard monitors aren't usually attractive to system crackers because they are only capable of harvesting passwords from users of the machine on which the monitor is installed. Where users have individual workstations, that boils down to one password per keyboard monitor. Only circumstances where many different people use the same keyboard (such as a courtesy workstation at a conference) offer sufficient payoff to a cracker to justify installation of a keyboard monitor. A separate solution for this circumstance is covered in Section 2.

### 1.2.2 The Real Problem: Hostile Sniffers

A "sniffer" is a privileged program running on a network-connected machine, listening to traffic on the attached network. For a sniffer to be effective, the network must function as a party line, that is to say messages to or from a given machine must also be available to other machines on the network. For a sniffer to be effective, it must be possible to make its network adapter listen to messages not addressed to it. Finally, in some circumstances, it may be necessary for the hardware network address of the adapter to be controllable by the attached machine. Sniffers are valuable tools for network and security administrators to use to diagnose activity on a network. Their installation is also a popular activity for crackers, once they have managed to run as the root user on a compromised machine.

Some of our networks seem not to be party-line networks. Certainly, the HiPPI network is a strictly point-to-point network, and it does not offer any machine the opportunity to observe messages being sent to other machines. Certainly, the classic coaxial Ethernet, and the more modern hub architecture are both party-line networks. So is FDDI. It is probably less well known that switched Ethernets must also be regarded as party-line networks. Switches are intended to send each packet only to the switch port to which it is addressed, however, while a switch is trying to determine the addresses of the Ethernet adapters connected to each of its ports, it broadcasts packets to all ports. It

is therefore only necessary to confuse a switch enough to put it into broadcast mode, converting it into a hub, in order to make a switched Ethernet a party-line network. If a cracker has privileged enough access to a machine to let him install a sniffer, he has the access required to convert a connected switch into a hub.

Promiscuous read mode is widely available on Ethernet and FDDI network adapters. In order to support multi-cast mode operation, software control over an adapter's hardware address is also widely available. With the ability to change an adapter's address comes the ability to flood a switch and turn it into a hub.

All the necessary technology is widely used. The sniffer software itself and the tools to install it are available from many sites over the Internet. Once a cracker has installed a sniffer he is able to harvest the passwords of all the users of all the machines on the attached subnet. The cracker's sniffer cost is about the same as his keyboard monitor cost but the payoff is much larger. That is why sniffers are a problem, and keyboard monitors generally are not.

## Chapter 2

# Solutions

The essential element of any solution is to give sniffers nothing of interest to look at. The hardware version of the solution is to make sure that the portion of the network that reaches each machine carries only packets intended for that machine. The software version of the solution is to authenticate using some technique other than cleartext, multiple-use passwords, and to encrypt any data channel that might carry a cleartext multiple-use password.

### 2.1 The Hardware Solutions

Two networking hardware technologies in use at the NAS can ensure that only network packets intended for a machine will be readable by that machine. HiPPI is a strictly switched network with static routing information embedded in its switches. Switched Ethernet is also capable of being configured to ensure that only packets intended for a machine can reach it. Both technologies have problems. The other major networking hardware technologies available are FDDI and Ethernet hubs, and they are both party-line technologies.

HiPPI is a relatively old, very high performance, low production volume, high-cost technology. Its switches need to be statically configured with routing information. Administration difficulty, overall cost, and particularly cost per client machine all conspire to make it very unlikely that our HiPPI network will be expanded enough to solve the sniffer problem.

Switched 100 megabit Ethernet can be configured to solve the problem. It is a new, high performance, high production volume, low-cost technology. By configuring it to drop any machine that changes its hardware Ethernet address, this technology can solve the sniffer problem. Unfortunately, by configuring switched Ethernet to drop machines that change their Ethernet addresses, one defeats multi-cast, a promising new networking technology, and increases the administrative load considerably.

The more popular configuration is to let the switch operate as a hub for any address it doesn't recognize, and to act as a switch for addresses for which

it has a port mapping. Port mappings are created by observing the Ethernet source addresses of incoming packets on each port. Our Ethernet switches are configured to detect new hardware Ethernet addresses, and to put them into the port translation tables. A compromised client of a switch need only flood it with enough Ethernet hardware address updates to flush other machines' entries from the port mapping table. That way, the switch will act as a hub and sniffing becomes possible.

Clearly, neither possible hardware solution is acceptable in our environment.

## 2.2 The Software Solutions

There are many available software solutions. Kerberos authenticates users both to local and to distant machines without ever trusting the networks involved. The S/Key protocol uses a different cleartext pass phrase for each authentication, so eavesdroppers only get pass phrases that will be useless to them in the future. Ssh uses public-key and private key cryptography to perform remote authentication and to protect communications channels that may carry passwords. These are only the most mature software solutions. There are many more.

### 2.2.1 Kerberos

Lack of proven scalability is the major problem with most of the newer possible software solutions. Though Kerberos is not new[26, 10], it too suffers from that problem. For Kerberos to work, each machine must be assigned a unique secret key. That key must also be known to a central Kerberos "ticket-granting" server. Because it has all the secret keys, the central server must be kept very secure. Only the Kerberos administrator should have access to it. That means that the Kerberos administrator has to assign machines their secret keys; nobody else can be permitted to get close enough to the central server to perform the task. It is also the case that each central server, and backup server(s), form a Kerberos "domain." Machines from different Kerberos domains can't authenticate users to each other unless their central servers are explicitly configured to do so. The limiting problem with Kerberos is scalability of administration.

### 2.2.2 S/Key

S/Key[7] hashes a user-provided hidden pass phrase to create a "first" usable pass phrase, and then hashes each usable pass phrase with the same algorithm to create the next. This pass phrase list creation takes place in advance of use, and under secure conditions. The creating machine only remembers the last pass phrase and its number within the sequence.

The actual list can be printed out for the user to carry around in wallet or purse. There are also S/Key calculators that can convert the hidden pass phrase and the sequence number of the desired usable pass phrase into the actual pass

phrase. Laptop computers, personal digital assistants and programmable pocket calculators all have S/Key calculator software available for them, often at no cost.

When the user tries to log into a machine that knows his “last” S/Key pass phrase and its sequence number, it will ask the user for the *previous* key in the sequence. The user can either enter it from his printed list, or calculate it from the hidden pass phrase. When the user enters the requested pass phrase, the machine runs the hash for one step and checks the result against the pass phrase it has been saving, the next one in the sequence. If the comparison is successful, the machine replaces the old pass phrase with the new one and reduces its remembered sequence number by one.

OPIE (One-time Passwords In Everything)[14] is a public domain software package that implements the S/Key protocol. It demands that password list creation be done the first time from the console of the S/Key-protected machine. Replacing the current list with a new (and longer) one can be done securely with OPIE at a distance. OPIE is obviously intended for use at a workstation. For our purposes, this is too restrictive. We need to modify OPIE to permit first-list creation within an enciphered session. Other than this one weakness, OPIE will serve well to allow people to log in securely from an insecure machine (such as a workstation in a workstation-room at a conference). Fortunately, OPIE is open-source software so the change can be made.

### 2.2.3 Secure Shell (Ssh)

Ssh replaces the functionality of the Berkeley Standard Distribution (BSD) Unix *r* commands, *rsh*, *rcp*, and *rlogin*, to permit, respectively, remote command execution, remote file transfer, and remote login sessions. It can also be stretched to support enciphering the command channel of the Internet standard File Transfer Protocol (FTP) programs. FTP performs all (password) authentication over its command channel and transfers data over a separate data channel.

Ssh is broadly configurable. It can be made to perform user authentication by transferring passwords over an enciphered channel, by authenticating the user's machine using public key cryptography and then accepting the user, based upon `/etc/hosts.equiv` or `~/.rhosts` files, or by directly authenticating the user using his own public key pair. There are several other variations.

The main problems with Ssh are legal and regulatory. The open source version (implementing Ssh protocol version 1.5) must be configured carefully and used carefully to remain on the right sides of intellectual property law and Federal regulations.

The main advantage of Ssh over Kerberos (for example) is that it can be made scalable from an administration point of view, at the cost of failure to achieve provable immunity to a Man in the Middle (MIM) attack. By contrast, there is no practical way to improve administration scalability for Kerberos. An attractive approach is to collect Ssh machine public keys randomly, and to investigate machine public key changes. Those two measures make a MIM attack very difficult and very easy to detect. The amount of administration effort can

then rise linearly with the number of machines administered and can be done by many different administrators. There is also no single central treasure chest. Ssh's valuable information is dispersed across all involved machines.

#### 2.2.4 The Secure Remote Password Protocol

Secure Remote Password (SRP) Protocol[30] is an open source authentication protocol. As a byproduct, it yields a secure shared session key that can be used to encipher session traffic. Reference implementations are available from Stanford University<sup>1</sup>. They come in Unix and Windows versions, but not in a MacOS version. They are implemented in C and Java, and support *telnet* and *ftp*, but they have no functionality like *rcp*, *rsh* or X11 port forwarding<sup>2</sup>. The author has made proposals to the Internet Engineering Task Force (IETF) and to the Institute of Electrical and Electronic Engineers (IEEE) P1363 Working Group.

SRP is promising, but immature. It also appears that the user's first introduction to each computer has to be face-to-face, in order to provide it with the necessary authenticating information. In this respect, it is similar to unmodified OPIE.

#### 2.2.5 Other Software Solutions

Other software solutions are available for at least some part of our problem. None of these solve enough of it to make the cut.

### OpenSSH

OpenSSH is a product of the OpenBSD<sup>3</sup> project. It is based upon Ssh version 1.2.12, which was the last completely unencumbered release of Ssh. It has

---

<sup>1</sup>SRP is available from its own Stanford University web site. <http://srp.stanford.edu/srp/>

<sup>2</sup>In this context "ports" are numbers used together with an IP address to specify one end of a TCP/IP communication channel. Ports come in privileged and unprivileged forms. Ports 1024 and above are unprivileged and can be acquired by programs running as an unprivileged user. Lower numbered ports can only be acquired by programs running as root.

Port forwarding is a capability of Ssh. If some program makes a TCP connection to a forwarded port, Ssh will connect to the IP address and port specified for the other end and do so from the far end of the active ssh command. The forwarded port is bidirectional and enciphered.

Ssh supports three kinds of port forwarding: authentication agent port forwarding, X11 port forwarding, and arbitrary port forwarding. Authentication agent port forwarding allows a distant Ssh daemon attempting to connect to an authentication agent to be put in touch with the authentication agent that is the parent of the user's shell on his workstation, eliminating the need to expose the user's secret RSA authentication key. X11 port forwarding similarly forwards connections on the distant machine back to the X11 server on the user's workstation. Finally, arbitrary ports can be forwarded from the user's workstation to the distant machine or visa versa.

<sup>3</sup>More information on OpenSSH and other aspects of the OpenBSD project are available at their web site. <http://www.openbsd.org/>

been upgraded to use Ssh protocol version 1.5 or version 2.0 (instead of version 1.3, as used by its Ssh parent). It includes many feature additions, and it should be treated as a less used alternative to Ssh 1.2.27. One of its features beyond Ssh 1.2.27 is incorporation of the S/Key protocol, a welcome feature on the bastion hosts.

Unfortunately for OpenSSH, the license encumbrances of Ssh 1.2.27 don't affect us. Also, OpenSSH is not as completely developed as is Ssh 1.2.27, but it comes standard on current releases of OpenBSD and FreeBSD, and possibly other free Unix systems as well.

## LSH

LSH<sup>4</sup> is a GNU<sup>5</sup> implementation of Ssh protocol version 2.0. As a GNU implementation, it overcomes the licensing problems of Ssh protocol version 2.0 for Unix, but it offers no help with MacOS or Windows, though it will work with F-Secure versions 2.x and above.

As was recently pointed out on the LSH web site, "LSH IS A WORK IN PROGRESS. DON'T EXPECT THE CURRENT VERSION TO WORK, AND \*DON'T\* EXPECT IT TO PROVIDE ANY SECURITY WHATSOEVER."

## OSSH

OSSH version 1.5.5<sup>6</sup> is a modified version of Ssh 1.2.12. Its main claim to fame seems to be that it supports the Andrew File System (AFS) and Kerberos IV authentication for Unix. It is being actively supported. It doesn't seem to have any advantages for us.

## Secure Socket Layer

The Secure Socket Layer (SSL) is the most commonly known of a family of protocols whose most recent member is called the Transport Layer Security (TLS) Protocol[4]. SSL versions 2.0 and 3.0 and TLS version 1.0 inter-operate. One widely available open source implementation is OpenSSL<sup>7</sup>.

While SSL is commonly used with web browsers for Internet commerce, there doesn't seem to be much in the way of remote session, remote file transfer, and X11 support outside the web browser world. SSL authentication on the web typically involves authenticating a web site to a browser implementation, not authenticating the user to a distant machine. SSL would also involve acquiring identity certificates for users from a Certificate Authority (CA).

---

<sup>4</sup>More information about LSH, including pointers to source code, is available at the LSH web site. <http://www.net.lut.ac.uk/psst/>

<sup>5</sup>GNU stands for "GNU's Not Unix." <http://www.gnu.org/>

<sup>6</sup>OSSH is available by FTP from Bjoern Groenvall's site. <ftp://ftp.pdc.kth.se/pub/krypto/ssh/>

<sup>7</sup>More information and source code are both available at the OpenSSL web site. <http://www.openssl.org/>

## Grid Security Infrastructure

Globus offers a secure *ftpd* and its coupled *ftp* utility and a set of modifications to Ssh 1.2.27 that all exploit its Grid Security Infrastructure (GSI)<sup>8</sup>. This is the direction we are going, with the Information Power Grid, but we're not there yet.

GSI requires a considerable infrastructure in addition to Ssh and *gsiftp* themselves. A CA is required to issue the certificates used with GSI. User access control lists must be maintained. The FTP daemon has to be replaced on machines to be reached by *gsiftp*. There is enough additional infrastructure required to use GSI that it is unattractive without Globus, and it is currently immature to boot. This is the direction the Information Power Grid is going. This makes a good future candidate for a parallel implementation as it matures.

## 2.3 The Real Solution - A Software Mix

The need to authenticate users to remote machines without the use of cleartext passwords can most easily be met in the NAS environment by a combination of two software solutions, OPIE and Ssh.

Ssh will cover all situations where the user can initially log onto a machine he can trust. That machine will contain Ssh software, the machine's own public key pair, and possibly the user's public and encrypted private keys. The user's encrypted private key, and the need to decrypt it is one reason why the machine must be worthy of trust. From this trusted machine, Ssh will let the user launch secure (enciphered) login sessions and file transfers, and cleartext file transfers (where performance is more important than the need to prevent eavesdropping on the data).

OPIE will cover situations where no trustworthy machine is available, but it is still necessary to authenticate the user to secure NAS machines without the risk of password theft. It will support insecure login sessions (where the content of the session can be the subject of eavesdropping) and cleartext file transfers. Authentication will be secure and there will be no opportunity for effective password theft.

It is important to realize that the choice of OPIE and Ssh in no way rules out parallel implementation of other secure authentication schemes. OPIE and Ssh are merely the minimum set of capabilities required to replace the function previously performed by cleartext passwords. Before choosing to implement parallel alternatives, though, one should consider the analogy of a walled city. Each authentication system is a guarded gate in the wall. More gates are more convenient than are fewer gates. Unfortunately, more gates are also harder to defend, and more gates stretch security efforts thinner than do fewer gates. Before adding more schemes, one should consider their effect upon the balance between convenience and security.

---

<sup>8</sup>For more details, consult the GSI web page. <http://www.globus.org/security/v1.1/index.htm##gsiftp>

## Chapter 3

# The Preferred Configuration

The combination of Ssh and OPIE necessary to protect passwords from sniffing is pretty straightforward. For Ssh itself it consists of a collection of required and prohibited capabilities. There is an OPIE modification element. One system configuration applies to Bouncer and Bruiser (the bastion hosts). Another configuration applies to all other Unix machines. Other configurations of Ssh and OPIE-enabled FTP clients apply to Macintoshes and to Windows machines.

### 3.1 Unix Ssh Do and Don't List

Unfortunately, cryptography is hot from political and property rights points of view, and Ssh uses cryptography. As a result, configuration decisions for Ssh must be made for non-technical reasons as well as for technical ones.

#### 3.1.1 Place Ssh Under /usr/prg, Not Under /usr/local

This may not apply to Unix machines that are not supported by the NAS Systems Division. NAS-supported Unix machines generally mount /usr/local from file servers, over the Network File System (NFS). NFS is notoriously insecure with respect to sniffers, so it makes no sense to expose sensitive Ssh files to the tender mercies of NFS. On any machine that remote mounts /usr/local, put Ssh somewhere else, like /usr/prg.

#### 3.1.2 Enable X Forwarding

This is an obvious requirement for Unix machines that support an X server. X forwarding provides an encrypted channel for any windows on the local server that were opened from a distant client that, in turn, was started through the services of Ssh. Somewhat less obvious is that X forwarding should be enabled

on Unix machines, such as von Neumann, that do not also support X servers. The reason is that a user may log in through the services of Ssh from an X-equipped workstation, and then use Ssh to launch an X client on a third machine. If X forwarding is not enabled on the middle machine, the window from the third machine may not be displayable on the first machine's X server, or worse, may not be encrypted in transit. It is important for remote X windows to be encrypted in transit, since the keyboard input sides of such windows can carry passwords (for *su* or *sudo* sessions, for example).

### 3.1.3 Disable the IDEA Cipher

The International Data Encryption Algorithm (IDEA)[11] is a technically sound cipher. It is fast. In spite of a good deal of trying to crack IDEA, the open literature[2, 3, 16] testifies that it is cryptanalytically robust. Unfortunately, intellectual property law makes it unattractive. IDEA has been patented[13, 12] by a Swiss firm. Getting a license for U.S. Government use is just too much of a bother, given that there is an alternative.

### 3.1.4 Enable DES

The Data Encryption Standard (DES)[19] is a technically unsound cipher. It is slow when implemented in software. Several papers[5, 8, 15, 29] and a book[6] have been published outlining ways to crack it efficiently. There would be no need to use it at all, except for regulations. NASA regulation NPG 2810.1[20, Section 4.11.2(e)1] requires that if a decision is made to encipher data, the DES "technique" (or 3DES) must be used "to protect data during telecommunications processes." It is therefore necessary to enable DES so users can specify it when they use *scp* to transfer data they believe should be enciphered in transit.

### 3.1.5 Enable 3DES

Triple DES (3DES)[19] is a technically sound cipher. 3DES amounts to applying the DES cipher to the cleartext, applying it again to the resulting ciphertext but using a different key, and repeating the process with a third key. This is the cipher of choice for those who worry that DES is too fast. There would be no need to use it at all, except for regulations. NASA regulation NPG 2810.1[20, Section 4.11.2(e)1] requires that if a decision is made to encipher data, the 3DES "technique" (or DES) must be used "to protect data during telecommunications processes." It is therefore necessary to enable 3DES so users can specify it when they use *scp* to transfer data they believe should be enciphered in transit.

### 3.1.6 Build Ssh With libwrap

Libwrap is a library that implements the TCP Wrapper[28]. TCP Wrapper controls incoming access to TCP and IP ports through a configuration file. It prevents access from certain hosts, or logs accesses from certain hosts, or

from whole classes of hosts. The desired actions are specified in each machine's configuration file. Building with libwrap allows Ssh's well-known port to be controlled even though *sshd* must run stand-alone<sup>1</sup>.

### 3.1.7 Build Ssh Without RSAREF on Government Machines

Ssh uses the RSA[22] public key algorithm. It is patented[23]. Fortunately for NASA, the research that resulted in the RSA algorithms was funded in part by the U.S. Government, and as a result, the Government has a free license<sup>2</sup>. Non Government-owned machines aren't covered. Therefore, if you build Ssh for a non Government-owned machine, you must use RSAREF. RSAREF is an RSA demonstration library, made freely available by the patent holders. It is slower than the RSA implementation included with Ssh, but is useful until September 20, 2000 to avoid patent infringement on other than Government machines.

If you must build with RSAREF, be aware that you will have to apply the patches<sup>3</sup> provided by the Computer Emergency Response Team (CERT).

### 3.1.8 Make Blowfish the Default Cipher

Blowfish[24] is a technically sound cipher. Its implementation and IDEA's are the two fastest in Ssh. Many cryptanalysts have tried and the open literature[25, 21, 27] chronicles nothing but failed attempts to crack it. For password protection, it is the clear winner. For regulatory reasons, though, users still ought to specify the *scp* option to use DES or 3DES when transferring data they believe should be enciphered in transit.

## 3.2 OPIE

OPIE will run on the bastion hosts, Bouncer and Bruiser. As it comes in the distribution<sup>4</sup>, OPIE requires that the user specify his first set of one-time passwords from the system console. That is not acceptable in our environment.

---

<sup>1</sup>TCP Wrapper works in one of two ways. Daemons that do not have to provide service too frequently, or that do not have an exorbitant start-up cost can be started on demand by *inetd*. TCP Wrapper can then exercise access control through *inetd*'s configuration file, */etc/inetd.conf*. Daemons that don't work well with *inetd* must be started independently, and must be permitted to run continuously. Independent daemons must be built with the libwrap library in order to be controllable by TCP Wrapper.

*Sshd* can't be started by *inetd* because it has to spend significant time computing a public key pair as its first order of business.

<sup>2</sup>"The Government has rights in this invention pursuant to Contract No. N00014-67-A-0204, awarded by the Department of the Navy, and Grant No. MCS76-14249, awarded by the National Science Foundation." [23, Government Interests section]

<sup>3</sup>The RSAREF patch is available directly from CERT/CC by FTP at <ftp://ftp.core-sdi.com/pub/patches/rsaref2.patch>, or by HTTP. <http://www.cert.org/advisories/CA-99-15/rsa-patch.txt> The Ssh patch is also available directly from CERT/CC. <http://www.cert.org/advisories/CA-99-15/ssh-patch.txt>

<sup>4</sup>OPIE version 2.32 is available from The Inner Net. <http://www.inner.net/pub/opie>

| Host Type  | URL   |
|------------|---|
| Unix       | None required: opiekey(1) is part of the OPIE package.  |
| Macintosh  | <a href="http://www.ja.net/CERT/Software/OPIE/contrib/macopie-1.1.hqx">http://www.ja.net/CERT/Software/OPIE/contrib/macopie-1.1.hqx</a>   |
|            | <a href="http://www.ja.net/CERT/Software/OPIE/contrib/OPIEcalc.sit.hqx">http://www.ja.net/CERT/Software/OPIE/contrib/OPIEcalc.sit.hqx</a> |
| Windows    | <a href="http://www.inner.net/pub/opie/contrib/WinKey21.exe">http://www.inner.net/pub/opie/contrib/WinKey21.exe</a>                       |
| tcl/tk     | <a href="http://www.inner.net/pub/opie/contrib/opie.tk-v2.3.gz">http://www.inner.net/pub/opie/contrib/opie.tk-v2.3.gz</a>                 |
| Palm Pilot | <a href="http://www.inner.net/pub/opie/contrib/pilOTP.zip">http://www.inner.net/pub/opie/contrib/pilOTP.zip</a>                           |
| HP-48      | <a href="http://www.inner.net/pub/opie/contrib/skey-hp48.tar.gz">http://www.inner.net/pub/opie/contrib/skey-hp48.tar.gz</a>               |

Table 3.1: Available OPIE calculator programs

Instead, OPIE must permit initial password set creation from an Ssh connection. There is no additional requirement for OPIE on Bouncer to honor a password list generated by OPIE on Bruiser, or vice versa. OPIE on the bastion hosts will support *ftpd*, *rlogind*, and *telnetd*. *Rlogind* and *telnetd* will permit people to authenticate their interactive sessions securely from locations that have no secure computers (such as conferences). Bouncer and Bruiser each have about 50 gigabytes of temporary space in their respective (and aptly named) */nobackup* file systems available for use by OPIE and *ftp*.

### 3.2.1 OPIE Calculators

OPIE calculators, such as those listed in Table 3.1 are a good alternative to a printed list of one-time pass phrases. Instead of having to refer to a printed list, a user who has an OPIE calculator need only enter his hidden pass phrase and the OPIE sequence number; the calculator then returns the currently required one-time pass phrase. References to to all known OPIE calculators will be placed on the NAS Computer Security web page as a user education measure.

## 3.3 FTP

There will be two different configurations of *ftpd* across the set of NAS machines. One configuration will apply to the bastion machines, Bouncer and Bruiser. The other will apply to all other NAS machines.

### 3.3.1 Standard NAS FTP Configuration

All NAS machines except the bastion hosts, Bouncer and Bruiser, will be configured to permit connection to their copies of *ftpd* only from their own IP addresses. This restriction will be accomplished using TCP Wrapper and is part of ensuring that the FTP command channel is enciphered. Each */etc/hosts.allow* file must accept connections from IP addresses corresponding to any actual network adapter attached to the machine. There is no harm in also allowing connections from localhost (127.0.0.1), though it can't be used with Ssh to protect the FTP command channel.

The reason for restricting *ftpd* access is to ensure that the command channel for any FTP session is not vulnerable to sniffing. Though an actual, cleartext connection from an FTP client on the local host will be possible, it won't be very useful, and it is not the intended use of the local connection. Local connections are intended to come from the local copy of *sshd*, as a result of Ssh port forwarding. Since the process can be a little tricky, port forwarding will most often be invoked by the FTP helper script discussed in Section 5.4, rather than directly by the user.

### 3.3.2 Bastion Host FTP Configuration

Bouncer and Bruiser will have to make FTP transfers to and from machines both inside and outside the NAS network. Inside transfers will be necessary to support large files whose security requirements include little concern for eavesdropping. Outside transfers will be necessary to support the file transfer needs of those who are connecting from an insecure machine. Therefore, on Bouncer and Bruiser only, connection to *ftpd* will be permitted from all IP addresses, and *ftpd* will be configured to use one-time passwords.

## 3.4 Macintosh Configuration

The NAS standard Macintosh configuration will be NiftyTelnet 1.1 SSH r3, a freeware product<sup>5</sup>. Now that the RSA patent[23] has expired, everyone can use NiftyTelnet without patent licensing problems. Other alternatives are:

- The commercial product, F-Secure<sup>6</sup> version 1.1, build 15 provides ssh session and remote shell service. *Do not* substitute F-Secure versions 2.x. That series of products uses incompatible protocol version~2.0. Versions~3.x of F-Secure are reputed to work with either protocol. The free software package, Fetch 3.0.3<sup>7</sup> provides OPIE and Ssh-assisted FTP file transfer (since *scp* support is missing from F-Secure 1.1). No OPIE calculator is needed because Fetch has one built in.
- BetterTelnet version 2.0fc1<sup>8</sup> offers features similar to F-Secure. The current release has finally replaced the buggy pre-release. Little is known about reliability. Fetch is also required with this choice.

---

<sup>5</sup>NiftyTelnet 1.1 SSH r3 is available from Jonas Wallden's web site. <http://www.lysator.liu.se/~jonasw/freeware/niftyssh/>

<sup>6</sup>F-Secure is available from F-Secure Corporation, formerly known as Data Fellows, Ltd. They are reachable at 675 N. First Street, Suite 605, San Jose, CA 95112. Their phone is (408) 938-6700, and they have a FAX at (408) 938-6701. They also have a web site. <http://www.fsecure.com/>

<sup>7</sup>Fetch is available from Dartmouth University. <http://www.dartmouth.edu/pages/softdev/fetch.html>

<sup>8</sup>BetterTelnet is available from Rolf Braun's web site. <http://www.cstone.net/~rbraun/mac/telnet/>

Macintosh users with NiftyTelnet who want the file transfer performance advantage of FTP over *scp* (and who don't mind that FTP gets that performance advantage by leaving their data in cleartext) should also have Fetch installed on their machines. Those who need Ssh secure port forwarding should also have F-Secure 1.1 installed, since NiftyTelnet doesn't support that feature. Neither F-Secure nor any other Ssh package for the Mac supports authentication port forwarding or authentication agents.

### 3.5 Windows 95/98/NT/2K Configuration

The NAS standard Windows configuration will come in two parts.

The Ssh element is the Tera Term (pro) SSH (TTSSH) extension Dynamic Link Library (DLL), version 1.5.1<sup>9</sup>. It is a public domain package that gives Tera Term (Pro), version 2.3<sup>10</sup>, Ssh-enhanced *telnet* capability. It supports RSA, RhostsRSA, and password authentication. It also supports X port forwarding and arbitrary port forwarding, making secure FTP possible in conjunction with LeechFTP. Since it is built using RSAREF, it causes no intellectual property rights problems on non-Government machines. Usage documentation is at its web site. TTSSH does not have an *scp* capability. It does not include any means of key pair creation, though it does accept keys created under Unix by *ssh-keygen*.

LeechFTP 1.3<sup>11</sup> provides FTP file transfer (since *scp* support is missing from TTSSH).

Other alternatives exist.

- The commercial product, F-Secure<sup>6</sup> version 1.1, build 15 has capabilities similar to TTSSH. It has its own key generation component, unlike TTSSH. Its port forwarding feature is picky about port numbers, making secure FTP set up slightly less convenient than with TTSSH.
- SecureCRT 3.0.3<sup>12</sup> is a commercial product that supports interactive Ssh connections under both Ssh protocol version 1.5 and version 2.0. It also supports X port forwarding and arbitrary port forwarding, making secure FTP possible in conjunction with LeechFTP. SecureCRT does not have an *scp* capability.

---

<sup>9</sup>TTSSH is available at Robert O'Callahan's web site. <http://www.zip.com.au/~roca/ttssh.html> The release files are signed by a PGP key for "Robert O'Callahan <roc+@cs.cmu.edu>", a 1024-bit key with the PGP fingerprint, 54 3D 18 28 5F 47 EA C2 B3 CC FB FB 8B 9B 26 53.

<sup>10</sup>Tera Term (Pro), version 2.3, is available at the Tera Term web site. <http://www.vector.co.jp/authors/VA002416/teraterm.html>

<sup>11</sup>LeechFTP is available directly from the author, Jan Debis, on his web page. <http://stud.fh-heilbronn.de/~jdebis/leechftp/>

<sup>12</sup>SecureCRT is a product of Van Dyke Technologies. It is available for download directly from the company's web site. <http://www.vandyke.com/>

- There is a straight Ssh 1.2.24 port<sup>13</sup> of the interactive login capability of *ssh*. There is little information available and the port has significant loose ends.
- PuTTY<sup>14</sup> is a public domain Ssh client package for Windows that offers both an Ssh enhanced *telnet* client and *scp* client functionality. Unfortunately, it has little documentation. Also, it does not use RSAREF, making it unsuitable for use on non-Government owned machines.

---

<sup>13</sup>The port is available at Gordon Chaffee's web site. <http://bmrc.berkeley.edu/people/chaffee/winntutil.html#sshnt>

<sup>14</sup>"It's called PuTTY partly because it makes Windows usable." PuTTY is available directly from Simon Tatham's web site. <http://www.chiark.greenend.org.uk/~sgtatham/putty/>

DRAFT

## Chapter 4

# The Usage Model

This chapter outlines the intended user experience that will result from the preferred configuration of Ssh, OPIE, the associated configuration files, and the helper scripts. The experience will differ with the user's situation. Users on NAS Unix machines and on non-NAS Unix machines they trust will have one sort of experience. Macintosh usage will be a bit different, as will Windows usage. Finally, users of untrusted machines (such as a workstation in a conference workstation room) will have yet another experience. Each aspect of the experience is handled separately.

### 4.1 General Issues

The following paragraphs cover general issues associated with each aspect of the usage model.

#### Introduction

The introduction problem is different for OPIE and for each Ssh authentication method. Three authentication methods will be available under the Ssh protocol, RhostsRSA, RSA, and password. Password authentication is the most straightforward and the least convenient, and Ssh tries it last. No introduction is required to use it. RSA with an authentication agent is the most convenient but requires some set-up by the user, and Ssh tries it second. RhostsRSA is mid-way between the other two in both subtlety and convenience, and Ssh tries it first.

For RSA authentication to work, the user must have made himself a personal key pair, and a copy of the public half must be stored in `~/.ssh/authorized_keys` on the destination Unix machine. For RhostsRSA authentication to work, the user's machine must have a public key pair, and a copy of the public half must be stored either in `etc/ssh_known_hosts`<sup>1</sup> or in `~/.ssh/known_hosts` on the desti-

---

<sup>1</sup>The exact leading portion of the path depends upon how Ssh is configured. Using the

nation Unix machine. On multi-user machines, public keys for other multi-user machines should be stored in `etc/ssh_known_hosts` to avoid repetitive storage in every user's `~/.ssh/known_hosts` file and to permit centralized updates.

For OPIE, the introduction problem is the problem of how to create the user's first set of one-time passwords. For Ssh, the introduction problem is the problem of how to store the required public keys in the right places. There is no introduction problem for password authentication, but there is one for RhostsRSA and RSA authentication methods.

### Machine Key Pair Creation

The machine key pair required for RhostsRSA authentication must be created before the user's machine can be introduced. This *must* be done once for each machine. Each machine's key pair must be unique.

### User Key Pair Creation

The user key pair required for RSA authentication must be created before the public half of it can be added to `~/.ssh/authorized_keys` on all machines the user wishes to contact using RSA authentication. This should be done once for each identity the user wants. Presuming he is not suffering from multiple personality disorder, one pair should suffice.

### Logging In Remotely

This is the equivalent of launching a *telnet* or *rlogin* session on the user's machine, with the target being a NAS Unix machine.

When logging in from outside the NAS network, if the NAS host's TCP Wrapper configuration forbids connections from non-NAS hosts, it may be necessary to "bounce" a connection through one of the bastion hosts, Bouncer or Bruiser. "Bouncing" means either logging in to the bastion and then logging in to the destination host from there, or it means executing the Ssh equivalent of the compound command, "*rsh bastion rlogin destination*".

### Remote Execution

This is the equivalent of launching an *rsh* command on the user's machine with the target being a NAS Unix machine.

"Bouncing" works the same way for remote execution as for logging in. One can either log into a bastion host and do remote execution from there, or execute the equivalent of the compound command, "*rsh bastion rsh destination command*".

---

recommended configuration for NAS machines, the full path name for `ssh_known_hosts` would be `/usr/prg/etc/ssh_known_hosts`. The Ssh default is `/usr/local/etc/ssh_known_hosts`. On machines that don't NFS mount `/usr/local`, the Ssh default is perfectly okay.

## Port Forwarding

There are three kinds of port forwarding, X11 port forwarding, Ssh authentication agent port forwarding, and arbitrary port forwarding (which is used to protect the FTP protocol's command channel). Port forwarding can be done in conjunction with an Ssh command to log in, or to execute a command remotely.

"Bouncing" an arbitrarily forwarded local port is tricky because the forwarded port can't be used until the second stage connection has been completed, but the compound command has to be executed in the background so the application that is going to use the port can run. By contrast, "bouncing" either an X11 port or an authentication agent port is easy.

## Enciphered File Transfer

This is the equivalent of launching an *rcp* command on the user's machine with the target being a NAS Unix machine. NASA regulations[20, Section 4.11.2(e)] require that the DES or 3DES ciphers be used if enciphering is necessary.

"Bouncing" works a little differently. It is necessary to remotely execute the file transfer command on the NAS host. That might produce a compound command equivalent to "*rsh bastion rsh destination rcp source sink*".

## Cleartext File Transfer

This is the equivalent of launching an *ftp* command on the user's machine with the target being a NAS Unix machine. The FTP protocol uses two connections, one for commands (and passwords) and the other for data. Although the data may be transferred in the clear, the passwords sent over the command channel must be protected.

"Bouncing" works differently for each user situation.

## Reversed Operations

Reversed operations are any of the previously discussed operations, but conducted from a NAS Unix machine and with the user's machine as the destination, instead of the other way around.

## 4.2 Trusted Unix Machines

This class of machine will be the easiest to use. Once it is properly set up, a NAS-supported or user-trusted Unix machine should be as convenient to use as a relatively insecure Unix machine. It should be capable of both initiating interactions with other machines and of being the target of an interaction launched from another machine. Individual issues are outlined below:

## Introduction

RhostsRSA introduction is handled automatically for Unix machines on the NAS network and for those additional Unix machines whose users have asked that they be included. These machines' Ssh daemons are queried randomly for their public machine keys, as outlined in Section 5.2.

Introduction of other user machines can be done automatically, simply by logging into them from a NAS machine, using

```
-o "StrictHostKeyChecking ask"
```

as an option, and responding, when asked, that the new host key should be accepted. Machines that have multiple names or IP addresses can be contacted multiple times. The user can also contact each machine once, and then manually edit the `~/.ssh/known_hosts` file so the first white-space delimited field of each machine's line contains a comma delimited list of all its fully qualified domain names, all its `/etc/resolv.conf` supported nicknames, and all its IP addresses.

RSA introduction must be done manually by the user.

## Machine Key Pair Creation

Generally, the Ssh build process will create a machine key pair if one doesn't already exist in `etc/ssh_host_key1` and `etc/ssh_host_key.pub`. `Ssh_host_key` must be owned by root and be readable only by root because it cannot be encrypted. It has to be in cleartext so `sshd` can read it when needed. If, for some reason, a machine must be given a new key pair, the `ssh-keygen` utility is available for the purpose.

Before manually adding a machine public key to a `known_hosts` file, two things should be done. First, the comment field should be removed. Second, a comma-separated list of all the machine's IP addresses and names (including abbreviated names as supported by `/etc/resolv.conf`) should be added at the beginning and set off with a blank.

## User Key Pair Creation

User key pair creation is not a part of the default Ssh build process. In order to take advantage of the Ssh authentication agent and RSA authentication, the user must create a personal Ssh key pair, by using `ssh-keygen`. The user must add the public key to the file, `~/.ssh/authorized_keys`, on any machine he wishes to use. The user must then install the private key, on every machine he logs into directly, as the file `~/.ssh/identity`. It is critically important that the user tell `ssh-keygen` to encrypt his private key, and that it be kept only in files with 0400 permission, owned by the user.

## Logging In Remotely

The `rlogin` helper script can be used, or the `ssh` utility can be used directly. There should be no request for the user's password because introduction should

have enabled RhostsRSA authentication.

The *rlogin* helper script will handle the necessary “bounce” to a hidden host without user intervention, once the user’s `~/.hidden.hosts` file is configured.

### Remote Execution

The *rsh* helper script can be used, or the *ssh* utility can be used directly. There should be no request for the user’s password because introduction should have enabled RhostsRSA authentication.

The *rsh* helper script will handle the necessary “bounce” to a hidden host without user intervention, once the user’s `~/.hidden.hosts` file is configured.

### Port Forwarding

The *rlogin* or *rsh* helper scripts can be used, or the *ssh* utility can be used directly. There should be no request for the user’s password because introduction should have enabled RhostsRSA authentication.

Both helper scripts will handle the necessary “bounce” to a hidden host without user intervention, once the user’s `~/.hidden.hosts` file is configured.

### Enciphered File Transfer

The *rcp* helper scripts can be used, or the *scp* utility can be used directly.

The *rcp* helper script will handle the necessary “bounce” to a hidden host without user intervention, once the user’s `~/.hidden.hosts` file is configured. Direct *scp* users will have to execute the compound command, “*ssh bastion ssh destination scp source sink*” in order to “bounce.”

### Cleartext File Transfer

The *ftp* helper script can be used, and is highly recommended, since direct usage of *ssh* and *ftp* utilities can be tricky. The Ssh enciphered port forwarding capability must be used to protect the FTP protocol’s command channel. This involves executing *ssh* in the background while executing *ftp* in the foreground.

The *ftp* helper script will handle the necessary “bounce” to a hidden host without user intervention, once the user’s `~/.hidden.hosts` file is configured. To “bounce,” direct users will have to put a compound *ssh* command in the background, run *ftp* in the foreground, and manage a race condition.

### Reversed Operations

All operations work equally well when reversed. Operations launched from hidden hosts, targeting the user’s machine don’t even require a “bounce.”

## 4.3 MacOS

While Macintoshes will not have all the capabilities of Unix machines in the cleartext password free environment, it is important that at least the basic capabilities to remotely log in and to transfer files be preserved. Additionally, as many more of the Unix machine's capabilities as possible should be supported. Individual issues are outlined below:

### Introduction

NiftyTelnet doesn't support RhostsRSA authentication, so machine introduction is meaningless. RSA introduction must be done manually by the user.

### Machine Key Pair Creation

NiftyTelnet doesn't support RhostsRSA authentication, so machine key pair manufacture wouldn't make much sense. It doesn't do it.

### User Key Pair Creation

NiftyTelnet cannot create user key pairs. It can accept a user key pair created on another machine. The MacOS version of F-Secure can do user key pair generation.

### Logging In Remotely

NiftyTelnet permits the user to log into any machine that is not hidden.

To reach a hidden host, the user must first log into a bastion and then log into the destination from the bastion.

### Remote Execution

NiftyTelnet permits the user to execute commands remotely on any machine that is not hidden.

To reach a hidden host, the user must first log into a bastion and then execute remote commands from the bastion.

### Port Forwarding

NiftyTelnet doesn't do port forwarding at all. The MacOS version of F-Secure does X11 port forwarding and arbitrary port forwarding, but it doesn't do authentication agent port forwarding. Neither NiftyTelnet nor F-Secure supports authentication port forwarding because they have no authentication agents and they don't support logging into MacOS from some other machine (which might have an agent). Users with a requirement for X11 or arbitrary port forwarding should consider adding the version of F-Secure recommended in Section 3.4 to their machines.

To reach a hidden host, the user must first log into a bastion and then execute remote commands from the bastion.

### **Enciphered File Transfer**

NiftyTelnet supports *scp*.

To exchange files with a hidden host, users will have to log into the bastion and use its */nobackup* file system as a staging area. *Scp* can be used for the transfer between the user's Mac and the bastion host. To do the transfer between the bastion and the hidden host, the user can log into the bastion and do the usual Unix command from there. As the last step, the user should delete the temporary file from */nobackup*.

### **Cleartext File Transfer**

Users will be able to do Unix cleartext file transfer between the NAS destination machine and the */nobackup* file system on a bastion host. OPIE-enabled *Fetch* can be used for the transfer between the user's Mac and the bastion. Finally, remote execution should be used to delete the temporary file from */nobackup*.

### **Reversed Operations**

NiftyTelnet doesn't support the equivalent of a Unix daemon, so there is nothing for a distant host to contact. Reversed operations are therefore impossible.

## **4.4 Windows 95/98/NT/2K**

While Windows machines will not have all the capabilities of Unix machines in the cleartext password free environment, it is important that at least the basic capabilities to remotely log in and to transfer files be preserved. Additionally, as many more of the Unix machine's capabilities as possible should be supported. Individual issues are outlined below:

### **Introduction**

TTSSH doesn't provide a version of *sshd*, so *RhostsRSA* introduction must be done manually by the user. *RSA* introduction must also be done manually by the user.

### **Machine Key Pair Creation**

TTSSH doesn't do machine key pair creation, but it will use a key pair created under Unix, by *ssh-keygen*. Unlike most other implementations of *Ssh*, TTSSH requires that the machine private key be encrypted (a reasonable measure under Windows 95/98).

### **User Key Pair Creation**

TTSSH does not create user key pairs. It can accept a user key pair created on another machine. If that other machine runs Unix, the ASCII NL character at the end of the public key file will have to be converted to a CR and NL sequence. The private key file is treated as a binary file, so end-of-line conversion is not required.

### **Logging In Remotely**

TTSSH will permit the user to log into any machine that is not hidden.

To reach a hidden host, the user must first log into a bastion and then log into the destination from the bastion.

### **Remote Execution**

TTSSH does not support remote execution (no equivalent of Unix *rsh*). Instead, one must use remote login.

To reach a hidden host, the user must first log into a bastion and then execute remote commands from the bastion.

### **Port Forwarding**

TTSSH supports both X11 port forwarding and arbitrary port forwarding. It does not support authentication port forwarding because it doesn't support an authentication agent and it doesn't support logging into Windows from some other machine (which might have an agent).

To reach a hidden host, the user must first log into a bastion and then execute remote commands from the bastion.

### **Enciphered File Transfer**

TTSSH doesn't support *scp*. Enciphered file transfer is therefore not supported from Windows.

### **Cleartext File Transfer**

TTSSH and LeechFTP can work together to support an enciphered FTP command channel and a cleartext data channel. The effect is to lodge machine selection with TTSSH, and to specify the same connection to LeechFTP all the time, regardless of the desired FTP server host.

The user's login preference under TTSSH must be set to include forwarding of local port 21 to the well known *ftpd* port (21) on the login destination machine. To reach a hidden host, the user must first log into a bastion and then execute remote commands from the bastion.

Once all the preferences are set up, it is only necessary to log into the FTP target machine before making the FTP connection, and to tell LeechFTP to connect to the local machine (but not to localhost) instead of connecting

to the remote machine. The LeechFTP bookmark feature is the easiest way to give it the required direction.

**Reversed Operations** TTSSH doesn't support the equivalent of a Unix daemon, so there is nothing for a distant host to contact. Reversed operations are therefore impossible.

## 4.5 Untrusted Machines

An untrusted machine is any machine not administered by someone the user knows and trusts. For purposes of this plan, the user is assumed to trust NAS system administration. Authentication from untrusted machines must be done through OPIE. The classic example of the untrusted machine is a courtesy workstation at a conference. Some colleague's machine in another system administration domain also fits the description. Individual issues are outlined below:

### Introduction

OPIE introduction consists of the user generating his first set of one-time pass phrases. That must be done in advance, must be done separately on each bastion host, and must be done from within an Ssh login session.

### Machine Key Pair Creation

This operation is meaningless in the context of an untrusted machine.

### User Key Pair Creation

This operation is meaningless in the context of an untrusted machine.

### Logging In Remotely

The user must log into a bastion first, using OPIE, then log into the destination host. The two-stage process is required whether the destination is hidden or not. The user will be warned at login that he must not use any utility requiring a password since the session is subject to eavesdropping. The Ssh login from the bastion to the destination host must use RhostsRSA authentication, as that is the only way to avoid exposing passwords in this circumstance. OPIE's password-sequence generating utility is immune to eavesdropping, so it can be used.

### Remote Execution

The user must log into a bastion first, using OPIE, then execute remote commands on the destination host, using RhostsRSA authentication.

### **Enciphered File Transfer**

OPIE doesn't encipher anything; it only accomplishes secure authentication over insecure channels, using an insecure local machine. Enciphered file transfer is therefore not supported from untrusted machines.

### **Cleartext File Transfer**

Users will be able to do Unix cleartext file transfer between the NAS destination machine and the /nobackup file system on a bastion. The user can simply use the FTP client program on the untrusted machine to transfer files between the untrusted machine and the /nobackup file system on a bastion. Finally, remote execution should be used to delete the temporary file from /nobackup.

### **Reversed Operations**

Only the daemons, *telnetd*, *rlogind*, *rexecd*, *rshd*, and *ftpd* are going to be disabled or restricted on NAS machines. The user client programs themselves, *telnet*, *rlogin*, *rexec*, *rsh*, and *ftp* will be untouched. They can all be used to contact untrusted machines. To do so will place the user's password on the untrusted machine at risk of eavesdropping, but that password is already at risk by virtue of being presented to an untrusted machine. The moral is don't use the same password both for a NAS machine and for an untrusted machine.

## Chapter 5

# The Tasks to be Performed

Many tasks must be completed before Ssh and OPIE can support users' needs alone, without cleartext passwords. The following sections cover them one at a time. The Gantt chart in Figure 5.7 on page 40 associates each task with its section number.

### 5.1 Install Ssh Protocol 1.5 Everywhere

Ssh utilities will have to be installed on all Unix, MacOS, and Windows machines. For Unix machines, and wherever else possible, Ssh daemons must also be installed. The latest release is 1.2.27.

Unix Ssh is available from the main distribution site in Finland<sup>1</sup>. It is open source software, usable by all. It supports all forms of Ssh access. Configure it. Build it. Install it.

Publicly available NiftyTelnet supports Ssh logins and enciphered file transfer on the Macintosh. Un-enciphered file transfer between the NAS bastion hosts (Bouncer and Bruiser) and MacOS can be done using NiftyTelnet along with Fetch 3.0.3. Fetch is also freely available. NiftyTelnet does not support port forwarding. The commercial product, F-Secure version 1.1, build 15 be added to provide that capability.

TTSSH on Windows supports Ssh logins and port forwarding. It has no *scp* work-alike. Un-enciphered file transfer between Windows and any NAS machine is handled by a combination of TTSSH and LeechFTP version 1.3, build 1.3.1.207. LeechFTP is freely available.

```

# Each ssh client configuration value is defined for a given target
# host by its first appearance across all configuration sources.

# First come configuration blocks for specific hosts; ...

# ... then we have configuration blocks for wild-carded hosts, ...

Host *.boeing.com
    Compression no

Host *.arc.nasa.gov
    Compression no

Host *.larc.nasa.gov
    Compression no

Host *.gsfc.nasa.gov
    Compression no

Host *.nas.nasa.gov
    Compression no

Host *.*
    Compression yes

# ... followed, ultimately, by the ultimate wild-carded host.
# Parameters that are allowed to default are listed as comments.

Host *
# BatchMode no
# Cipher blowfish
# ClearAllForwardings yes
# Compression no
# CompressionLevel 6
# ConnectionAttempts 1
# EscapeChar ~
# FallBackToRsh yes
# ForwardAgent yes
# ForwardX11 yes
# GatewayPorts yes
# GlobalKnownHostsFile /usr/local/etc/ssh_known_hosts
# HostName *
# IdentityFile ~/.ssh/identity
# KeepAlive yes
# KerberosAuthentication no
# KerberosTgtPassing no
# LocalForward
# NumberOfPasswordPrompts 1
# PasswordAuthentication yes
# PasswordPromptHost yes
# PasswordPromptLogin yes
# Port 22
# ProxyCommand /usr/local/bin/ssh -p %p %h
# RemoteForward
# RhostsAuthentication yes
# RhostsRSAAuthentication yes
# RSAAuthentication yes
# StrictHostKeyChecking yes
# TISAuthentication no
# UsePrivilegedPort yes
# User
# UserKnownHostsFile ~/.ssh/known_hosts
# UseRsh no
# XAuthLocation /usr/X11R6/bin/xauth

```

Figure 5.1: Model Ssh system-wide client configuration file

```

# This is the Ssh daemon systemwide configuration file. All
# parameters are listed in alphabetical order. Parameters that are
# allowed to default are listed as comments.

# The philosophy of this configuration is that the client process
# cares only about fulfilling the user's request and the server is
# responsible for maintaining security.

# Example:
#   The client configuration file permits Rhosts authentication but
#   the server configuration file forbids it.

# AllowGroups *
# AllowHosts *
# AccountExpireWarningDays 14
# AllowSHosts *
# AllowTcpForwarding yes
# AllowUsers *
# CheckMail yes
# DenyGroups
# DenyHosts
# DenySHosts
# DenyUsers
# FascistLogging no
# ForcedEmptyPasswdChange no
# ForcedPasswdChange yes
# HostKey /usr/local/etc/ssh_host_key
# IdleTimeouttime
# IgnoreRhosts no
# IgnoreRootRhosts no
# KeepAlive yes
# KerberosAuthentication no
# KerberosOrLocalPasswd no
# KerberosTgtPassing no
# KeyRegenerationInterval 3600
# ListenAddress 0.0.0.0
# LoginGraceTime 600
# PasswordAuthentication yes
# PasswordExpireWarningDays 14
# PermitEmptyPasswords no
# PermitRootLogin no
# PidFile /var/run/sshd.pid
# Port 22
# PrintMotd yes
# QuietMode no
# RandomSeed /usr/local/etc/ssh_random_seed
# RhostsAuthentication no
# RhostsRSAAuthentication yes
# RSAAuthentication yes
# ServerKeyBits 768
# SilentDeny no
# StrictModes yes
# SyslogFacility AUTH
# TISAuthentication no
# Umask 077
# X11Forwarding yes
# X11DisplayOffset 10
# XauthLocation /usr/X11R6/bin/xauth

```

Figure 5.2: Model Ssh server daemon configuration file

```
bouncer.nas.nasa.gov
bruiser.nas.nasa.gov
```

Figure 5.3: Model Ssh shosts.equiv file.

```
#!/bin/sh
# $XConsortium: Xsession /main/10 1995/12/18 18:21:28 gildea $

# redirect errors to a file in user's home directory if we can
for errfile in "$HOME/.xsession-errors" "${TMPDIR-/tmp}/xses-$USER" "/tmp/xses-$USER"
do
    if ( cp /dev/null "$errfile" 2> /dev/null )
    then
        chmod 600 "$errfile"
        exec > "$errfile" 2>&1
        break
    fi
done

case $# in
1)
    case $1 in
    failsafe)
        exec ssh-agent xterm -geometry 80x24-0-0
        ;;
    esac
esac

startup=$HOME/.xsession
resources=$HOME/.Xresources

if [ -f "$startup" ]; then
    exec ssh-agent "$startup"
else
    if [ -f "$resources" ]; then
        xrdp -load "$resources"
    fi
    exec ssh-agent xsm
fi
```

Figure 5.4: Modified Xdm session start-up file

| <i>Option</i>     | <i>Reason</i>                                  |
|-------------------|--|
| --prefix=/usr/prg | Avoids NFS-mounted file systems                |
| --with-x          | Enables X11 forwarding                         |
| --without-idea    | IDEA has intellectual property rights problems |
| --with-des        | Required by regulation for data encryption     |
| --with-libwrap    | Incorporates TCP wrappers                      |
| --without-rsaref  | The RSA patent has expired                     |

Table 5.1: Required Unix Ssh Configuration Options

```
#
# If we are logging in from a secure port or through ssh, and no ancestor
# is an ssh-agent, morph into ssh-agent and spawn another login shell.
#
if (! $SSH_AUTH_SOCK) then
  /usr/bin/tty | /usr/bin/grep -q 'tty[pq]'
  if ($status || $SSH_CLIENT) then
    if (-x /usr/local/bin/ssh-agent) then
      exec /usr/local/bin/ssh-agent $SHELL -l
    endif
    echo '*****'
    echo '*   WARNING: SSH-AGENT WON'T START; SSH MAY BE DAMAGED   *'
    echo '*****'
  else
    echo '*****'
    echo '*   WARNING: INSECURE CONNECTION; USE NO PASSWORDS   *'
    echo '*****'
  endif
endif
```

Figure 5.5: Addition to system C-Shell startup

```

#
# If we are logging in from a secure port or through ssh, and no ancestor
# is an ssh-agent, morph into ssh-agent and spawn another login shell.
#
if [ -z "$SSH_AUTH_SOCK" ]; then
    /usr/bin/tty | /usr/bin/grep -qv 'tty[pq]'
    if [ $? -eq 0 -o -n "$SSH_CLIENT" ]; then
        [ -x /usr/local/bin/ssh-agent ] && \
        exec /usr/local/bin/ssh-agent $SHELL
        echo '*****'
        echo "*   WARNING: SSH-AGENT WON'T START; SSH MAY BE DAMAGED   *"
        echo '*****'
    else
        echo '*****'
        echo "*   WARNING: INSECURE CONNECTION; USE NO PASSWORDS   *"
        echo '*****'
    fi
fi

```

Figure 5.6: Addition to system Bourne Shell startup

### 5.1.1 Unix Configuration

Ssh Unix configuration issues fall into several categories. Installation location is important, particularly on machines that use NFS. Installation options are important, for reasons of proper function, as well as for legal and regulatory reasons. Configuration options are important so the whole package will work well across the NAS. Finally, Ssh can be made more user friendly if its own configuration files are modified.

There are many configuration options one can specify when building Ssh. For technical, regulatory, and legal reasons, the defaults are not always the right choice. The non-default options shown in Table 5.1 should be used.

A model Ssh client utility configuration file, `etc/ssh_config`<sup>2</sup>, is shown in Figure 5.1. It may need to be customized to each host. The “Host” blocks, for example, are appropriate to machines on the NAS net, but not necessarily to others. A model Ssh daemon configuration file, `etc/sshd_config`, is shown in Figure 5.2. It too may need to be customized to each host. The value of `GlobalKnownHostsFile`, for example, may have to be changed from the Ssh default. A model Ssh host equivalency file, `etc/shosts.equiv`, is shown in Figure 5.3. It is appropriate only for NAS systems other than the bastion hosts. Its purpose is to insure that users who log into a bastion host using OPIE (which leaves their session in clear text) are not then asked for a password when they log into other NAS machines. It ensures that such subsequent logins over Ssh will use

<sup>1</sup>The file is to be found at `ftp.funet.fi`. `ftp://ftp.funet.fi/pub/unix/security/login/ssh-1.2.27.tar.gz` Its MD5 checksum is `c22bc000bee0f7d6f4845eab72a81395`. There is a separate PGP signature file from “Ssh distribution key <ylo@cs.hut.fi>,” a 1024-bit key with the PGP fingerprint, `C8 90 C8 5A 08 F0 F5 FD 61 AF E6 FF CF D4 29 D9`.

<sup>2</sup>The leading part of the path name is uncertain. See footnote 1 on page 19.

RhostsRSA authentication.

Some non-Ssh configuration file changes are also appropriate for machines that have had Ssh installed. One type of change enables usage of *ssh-agent* for RSA authentication. The other lets the user know when his session is not protected by encryption.

The Ssh documentation says that *ssh-agent* does not have to be the parent of a user's session. Don't believe it. The alternative to parenthood is for the user's session to start a copy of *ssh-agent* at the beginning (say in `~/.cshrc`), and then to kill it as the session dies (say in `~/.logout`). This sort of measure doesn't always work, and can therefore cause a slow build-up of orphaned copies of *ssh-agent*. One should make *ssh-agent* the parent of each user's session. To do that, changes have to be made to the Xdm user session start-up file, `X11R6/lib/X11/xdm/Xsession`, and to the system-wide C-Shell and Bourne Shell start-up scripts.

The basic change that must be made to Xsession is to find all instances of *exec* and replace them with "*exec ssh-agent*." That has already been done in Figure 5.4. For systems that don't use Xsession, a shell script that *execs* *ssh-agent* to run the effective session client will be required. For the system-wide shell start-up scripts, things are a bit more complicated. It is important not to start *ssh-agent* if the user is connected through an insecure port. If the user's connection is insecure, these start-up script additions warn him. If secure, they turn the current shell into *ssh-agent* and start a new shell. The C-Shell version is shown in Figure 5.5, and the Bourne Shell version is shown in Figure 5.6. Some modification of the examples will probably be required. They were written and tested under FreeBSD, and such things as the test whether the controlling terminal is a pseudo tty will probably have to be customized.

**Status:** Not yet complete. See the schedule.

### 5.1.2 Personal Computer Configuration

The appropriate software must be installed on all NAS supported personal computers not already configured as described in Sections 3.4 and 3.5.

**Status:** Not yet complete. See the schedule.

## 5.2 Collect Public Machine Keys Automatically

In any public key cryptography system, key distribution integrity is a problem. You can't just accept keys as offered, or someone will be able to masquerade as someone else, merely by offering a public key that claims to be from the victim.

It is also true that when Ssh becomes the only means of connecting to NAS machines, the amount of disk space on multi-user machines devoted to the public keys of other multi-user machines can grow as  $mn^2$  where there are  $m$  users and  $n$  multi-user machines, if all public machine keys go into `~/.ssh/known_hosts`.

The solution to both problems is for NAS automatically to maintain the master host public keys file, `etc/ssh_known_hosts`<sup>2</sup>.

The necessary utility must:

1. Query the Ssh daemons on all machines on all NAS subnets plus the daemons on those machines listed on scientific users' account request forms as the Unix machines from which they would make first Ssh contact with NAS,
2. Begin each session of queries at random times,
3. Perform the queries in random order, and
4. Leave the resulting file of Ssh machine public keys in a central location<sup>3</sup>.

The file is necessary both to provide secure collection of public keys, and to prevent an explosion of entries in every user's `~/.ssh/known_hosts` file. Queries must be done randomly to make it as difficult as possible for a cracker to control a victim's IP address long enough to respond to us with a bogus public key for that IP address.

**Status:** The Local Area Network Group has completed this task, but see Section 5.3.

### 5.3 Check Users' Ssh Machine List

The first attempt to get a list of machines specified on scientific users' account request forms made it clear that a significant fraction of resulting database entries weren't useful. In some cases it seemed that the person filling out the account request form didn't get the idea and entered some general description instead of a machine name. In other cases, there could have been a typographical error while transferring information from the forms to the data base. The original list also contained only entries from account request forms received at the Next Operational Period (NOP) boundary, in the fall of 1999. Whatever the exact nature of the inaccuracy, all accounts with invalid information need follow-up, as do accounts approved since NOP.

**Status:** Not yet complete. See the schedule.

### 5.4 Distribute Helper Scripts

For those who are used to *rlogin*, *rsh*, *rcp*, and *ftp*, use of the *ssh* and *scp* utilities can seem alien. Worse, use of *ftp* through an enciphered Ssh command channel is tricky. These inconveniences and difficulties only become worse when

---

<sup>3</sup>Currently, that central location is on `marcy.nas.nasa.gov` in a file named `/usr/nas/generic/security/etc/ssh_known_hosts`.

it is necessary to “bounce” a connection through a bastion host. To make life easier for the users at no loss of security, scripts will be written that accept *r* command options (and *ftp* options) as well as appropriate Ssh options. They will also allow automatic bastion host handling, so the users will not have to contend with either inconvenience. Naturally, direct Ssh usage will also be possible.

When the helper script package is complete, it will be made available through the NAS Computer Security web page. It may also be installed in `/usr/nas/bin`. On a machine for which that is not possible, a user can still install the package in his own bin directory, and put that directory first on his search path.

**Status:** Not yet complete. See the schedule.

## 5.5 Install Modified OPIE on Bastions

OPIE has one serious problem for NAS use: it requires that the user create his initial set of one-time pass phrases from the system console on the authenticating machine, Bruiser or Bouncer in our case. It is unacceptable to have to usher users into room 133 to permit them to create their first set of one-time passwords. OPIE must be modified to accept an Ssh connected session as being secure enough to permit key generation. That will require a software modification.

The modifications will first be the subject of a code review. The code changes will then be made and OPIE will be built on Marcy. The first installation and tests will be done on Bruiser. After a decent interval of successful use by staff, the modified software will be installed on Bouncer, and will be advertised to the users.

**Status:** Not yet complete. See the schedule.

## 5.6 Educate Users

User education will take several forms. There will be NAS Computer Security web page upgrades. There will be a class aimed at workstation system administrators. There will be user familiarization classes.

The NAS Computer Security web page will get several additions:

- A browsable version of this plan will be available.
- There will be up-to-date how-to-get-it references for Unix, Mac and Windows versions of Ssh.
- It will be possible to download a copy of the helper scripts.
- Copies of slides for all classes will be available.
- There seems to be only one flawed tutorial book[1] on the market covering Ssh. Nonetheless, the web page should identify it.

A presentation should be made, explaining the configuration changes outlined in Section 5.1.1, for the benefit of Unix system administrators. Three more presentations for users are required. The first should outline the basics of Ssh usage. The second should cover installation and use of the helper scripts. The third should explain OPIE usage.

**Status:** Not yet complete. See the schedule.

## 5.7 Make Administration Utilities Work With Ssh

Any NAS administration tools that depend upon cleartext passwords, or the BSD *r* commands must be changed to work with Ssh and to work without *rlogind*, *telnetd*, *rshd*, *rexecd*, and direct access to *ftpd*.

### 5.7.1 PCP Modification for Ssh

Piped CoPy (PCP) authentication now involves */etc/hosts.equiv* files. Some modifications were made to it in the past to make it ready to use Ssh authentication instead. The task needs to be completed.

**Status:** Not yet complete. See the schedule.

### 5.7.2 PBS Configuration for Ssh

The Portable Batch System (PBS) has been capable of transferring files through the services of *rcp*, or through *scp* with RhostsRSA authentication since release 1.1.12. Before turning off access to *rcp*, it is important to make sure that all PBS servers across all NAS-supported machines are using *scp* and not *rcp*.

**Status:** Not yet complete. See the schedule.

## 5.8 Disable Cleartext Passwords

When all other tasks have been completed, actually turning off access by cleartext password will be a very small task. It consists of five steps conducted on all supported Unix machines:

1. Make sure that the remote login daemon, *rlogind*, is disabled and will not run. Preferably, its executable file should be removed.
2. Make sure that the remote terminal daemon, *telnetd*, is disabled and will not run. Preferably, its executable file should be removed.
3. Make sure that the remote shell daemon, *rshd*, is disabled and will not run. Preferably, its executable file should be removed.

4. Make sure that the remote execution daemon, *rexecd*, is disabled and will not run. Preferably, its executable file should be removed.
5. Configure TCP Wrapper to permit access to the FTP daemon, *ftpd*, only from the local machine's actual network IP addresses.
6. Configure NAS Computer Security scans to detect and report service at the well-known ports for *rlogind*, *telnetd*, *rshd*, *rexecd*, and *ftpd*.

That is all that needs to be done as the last step.

**Status:** Not yet complete. See the schedule.

## 5.9 Schedule

The schedule for the succession of tasks leading up to the elimination of cleartext passwords is shown in Figure 5.7. Major tasks are marked with the numbers of the sections that explain them.

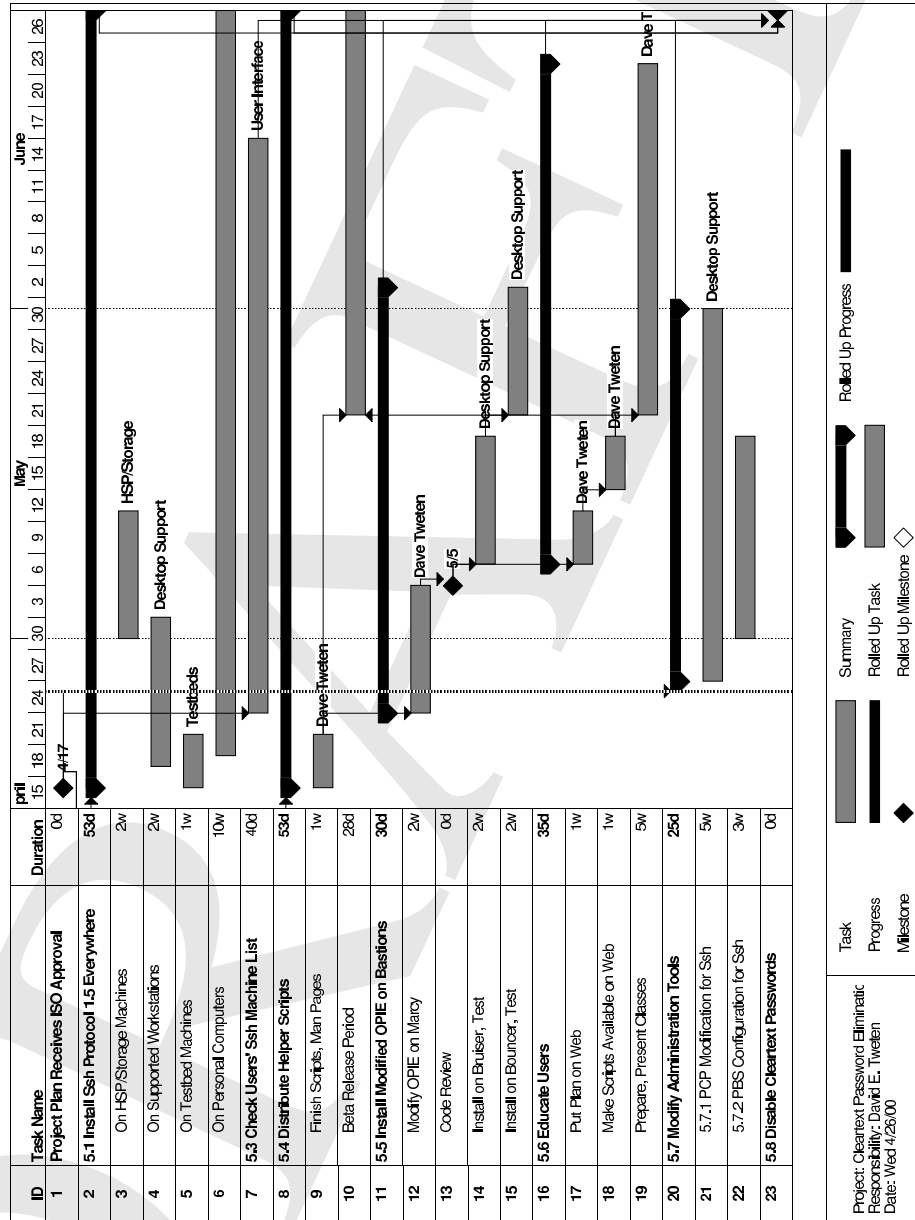


Figure 5.7: Schedule

# Appendix A

## ISO 9001 Considerations

Cleartext password elimination involves the delivery of software to NAS Systems division customers. ISO 9001 requirements therefore come into play.

### A.1 Software Project Characterization

The Qualitative method[9, Section 6.2.1] of characterization is the one used for this project. It classifies a project along three dimensions, risk, cost, and organizational complexity. Cleartext password elimination is a *Low Risk, Very Low Cost, Simple* project. The rationale for this classification is outlined below.

#### A.1.1 Software Project Risk Classification

By the Qualitative method, software must be placed into one of four risk categories[9, Table 1]. *High Risk* software can kill, maim or cause significant property damage if it fails. *Medium Risk* software can cause significant property damage or embarrassment to NASA by failing. Failure of *Low Risk* software will only cause lost user time or lost computer time, or may take significant time to fix. *Negligible Risk* software is likely to cause little lost user or computer time if it fails.

This project to eliminate the use of cleartext passwords with NAS computers is *Low Risk*. There is no possibility of injury or property damage that could flow from the project, and the only potential for embarrassment to NASA would be a high-profile computer security compromise, which is no more likely with this project than it would be without — almost certainly less likely.

#### A.1.2 Software Project Cost Classification

There are six potential cost categories[9, Table 2] into which a project can be placed: *Extremely High Cost* (whose budget requires higher than Center Director approval), *Very High Cost* (whose budget must be approved by the Center Director), *High Cost* (whose budget requires Directorate approval), *Medium*

*Cost* (requiring a Division Chief's budget approval), *Low Cost* (requiring the approval of a responsible manager below Division Chief), and *Very Low Cost* (whose budget is within the discretion of the software project manager).

This project to eliminate the use of cleartext passwords is *Very Low Cost*. It does not require the outright purchase of anything. It requires the full-time efforts of the software project manager for a couple of months, and intermittent cooperation from others in the Division over the same few months.

### A.1.3 Organizational Complexity Classification

There are three potential complexity categories[9, Table 3] into which a project can be placed: *Very Complex* (for projects involving several organizations or several locations), *Complex* (for projects involving no more than two organizations at no more than two locations), and *Simple* (for projects involving only a single organization at a single location).

The project to eliminate the use of cleartext passwords is *Simple*. The sole developing organization is the NAS Systems Division and the sole locus of development is NASA Ames Research Center.

## A.2 Selected Development Controls

Based upon the project characterization as *Low Risk*, *Very Low Cost*, and *Simple*, the following project controls were selected.

### A.2.1 Development Approach

The chosen development approach[17, Section 6.3] is *Single-Release Life Cycle*. The user interfaces are well understood and there is little likelihood of demand for additional features requiring subsequent releases.

### A.2.2 Documentation Plan

There will be an installation guide[17, Section 6.4.1] provided with the helper scripts (in the form of a README file). The helper scripts are the only parts of this project that are to be developed from scratch. Installation guides are already available for the project deliverables that are only to be configured or that are to be modified.

Design of the helper scripts will be documented[17, Section 6.4.2] through comment lines in the code. There will also be a Unix man page for each script. The requirements are outlined in Chapter 2 of this plan.

### A.2.3 Maintenance Plan

Software maintenance for the OPIE modification and for the helper scripts will be provided by the author as problems are identified on the e-mail lists, `ssh-admin@nas.nasa.gov` and `ssh-users@nas.nasa.gov`. Maintenance of the

configuration changes required will be provided by the system administrators involved. Maintenance for the non-source freeware components of the project will be provided through request to the authors.

#### **A.2.4 Software Configuration Control**

A master copy of each locally developed or open-source component of this project, and a machine-readable copy of this document will be maintained on the NAS Systems Division mass storage machine<sup>1</sup>, in the directory structure under `~tweten/Ssh_Project`.

#### **A.2.5 Change Request Tracking**

Change request tracking will be done informally. Users will be invited to send requests to one of the mailing lists, `ssh-admin@nas.nasa.gov` or `ssh-users@nas.nasa.gov`.

#### **A.2.6 Required Reviews**

The Project Plan review and the Requirements review will be conducted informally by Line Management[17, Section 6.8.1]. The combined attendee list and minutes of this review will be treated as a Quality Record[18].

#### **A.2.7 Replication, Delivery, and Installation**

The helper scripts will be available from the NAS Computer Security web page. That page will also contain references to all open-source and free components. Any commercial alternatives to the open-source or free components identified in this document that are desired by any user organization will be the financial responsibility of that organization. Other pieces, such as the OPIE modifications will not actually be delivered to customers.

---

<sup>1</sup>Lou.nas.nasa.gov, as of this writing.

DRAFT

# Bibliography

- [1] Anne Carasik. *UNIX Secure Shell*. McGraw-Hill, 1999.
- [2] Joan Daemen, René Govaerts, and Joos Vandewalle. A hardware design model for cryptographic algorithms. In *ESORICS 92, Proceedings of the Second European Symposium on Research in Computer Security*, pages 419–434. Springer-Verlag, 1992.
- [3] Joan Daemen, René Govaerts, and Joos Vandewalle. Weak keys for IDEA. In Douglas R. Stinson, editor, *Advances in Cryptology—CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 224–231, Santa Barbara, CA, August 22–26, 1993. Springer-Verlag, 1994.
- [4] Tim Dierks, Philip L. Karlton, Alan O. Freier, and Paul C. Kocher. The TLS protocol version 1.0. RFC 2246, Internet Engineering Task Force, January 1999. Available as <http://www.ietf.org/rfc/rfc2246.txt>.
- [5] Whitfield Diffie and Martin E. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, June 1977.
- [6] *Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design*. Electronic Frontier Foundation, San Francisco, 1998.
- [7] Neil Haller. The S/KEY one-time password system. RFC 1760, Internet Engineering Task Force, February 1995. Available as <http://www.ietf.org/rfc/rfc1760.txt>.
- [8] Martin E. Hellman. DES will be totally insecure within ten years. *IEEE Spectrum*, 16(7):32–39, July 1979.
- [9] W. Henry. Project management for the design, development, and maintenance of software. SLP 53.ARC.0004.1, NASA Ames Research Center, May 18, 1999. Available as [http://ace.arc.nasa.gov/cgi-bin/postdoc/get?url\\_id=7132&ext=pdf](http://ace.arc.nasa.gov/cgi-bin/postdoc/get?url_id=7132&ext=pdf).
- [10] John T. Kohl, B. Clifford Neuman, and Theodore Y. Ts'o. The evolution of the Kerberos authentication service. In *EurOpen Conference Proceedings*, pages 295–313, Tromsø, Norway, May 1991.

- [11] Xuejia Lai. *On the Design and Security of Block Ciphers*, volume 1 of *ETH Series in Information Processing*. Hartung-Gorre Verlag, Konstanz, 1992.
- [12] James L. Massey and Xuejia Lai. Device for the conversion of a digital block and use of same. U.S. Patent #5,214,703, May 25, 1993.
- [13] James Lee Massey and Xuejia Lai. Device for converting a digital block and the use thereof. International Patent PCT/CH91/00117, November 28, 1991.
- [14] Daniel L. McDonald, Randall J. Atkinson, and Craig Metz. One time passwords in everything (OPIE): Experiences with building and using stronger authentication. In *Fifth USENIX UNIX Security Symposium*, pages 177–186, Salt Lake City, UT, June 5–7, 1995. USENIX Association. Also available as <http://www.usenix.org/publications/library/proceedings/security95/mcdonald.html>.
- [15] Robert McLaughlin. Yet another machine to break DES. *Cryptologia*, XVI(2):136–144, April 1992.
- [16] Willi Meier. On the security of the IDEA block cipher. In Tor Helleseth, editor, *Advances in Cryptology—EUROCRYPT '93, Workshop on the Theory and Application of Cryptographic Techniques*, volume 765 of *Lecture Notes in Computer Science*, pages 371–385, Lofthus, Norway, May 23–27, 1993. Springer-Verlag, 1994.
- [17] G. Miyahara. Guidelines for implementing 43.ARC.0004.1. SLWI 53.ARC.0004.1.1, NASA Ames Research Center, November 29, 1999. Available as [http://ace.arc.nasa.gov/cgi-bin/postdoc/get?url\\_id=10704&ext=pdf](http://ace.arc.nasa.gov/cgi-bin/postdoc/get?url_id=10704&ext=pdf).
- [18] G. Miyahara. Quality records. SLP 53.ARC.0016, NASA Ames Research Center, June 8, 1999. Available as [http://ace.arc.nasa.gov/cgi-bin/postdoc/get?url\\_id=5554&ext=pdf](http://ace.arc.nasa.gov/cgi-bin/postdoc/get?url_id=5554&ext=pdf).
- [19] Data encryption standard. FIPS PUB 46-3, U.S. Department of Commerce, October 25, 1999. Also available as <http://csrc.nist.gov/fips/fips46-3.pdf>.
- [20] AO/Chief Information Officer. Security of information technology. NPG 2810.1, National Aeronautics and Space Administration, August 26, 1999. Available as [http://nodis.hq.nasa.gov/Library/Directives/NASA-WIDE/Procedures/Legal\\_Policies/N\\_PG\\_2810\\_1.html](http://nodis.hq.nasa.gov/Library/Directives/NASA-WIDE/Procedures/Legal_Policies/N_PG_2810_1.html).
- [21] Vincent Rijmen. *Cryptanalysis and Design of Iterated Block Ciphers*. PhD thesis, Katholieke Universiteit Leuven, Belgium, October 1997. Available through <http://www.esat.kuleuven.ac.be/~rijmen/pub97.html>.

- [22] R. L. Rivest, A. Shamir, and L. Adleman. A method of obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.
- [23] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. Cryptographic communications system and method. U.S. Patent #4,405,829, September 20, 1983.
- [24] B. Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In R. Anderson, editor, *Fast Software Encryption, Cambridge Security Workshop Proceedings*, number 809 in Lecture Notes in Computer Science, pages 191–204, Cambridge, UK, 1994. Springer-Verlag. Also available as <http://www.counterpane.com/bfsverlag.html>.
- [25] B. Schneier. The blowfish encryption algorithm—one year later. *Dr. Dobbs Journal*, September 1995. Also available as <http://www.counterpane.com/bfdobsoyl.html>.
- [26] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter Conference*, pages 191–202, Dallas, TX, January 1988. USENIX Association. Also available as <ftp://athena-dist.mit.edu/pub/kerberos/doc/usenix.PS>.
- [27] Serge Vaudenay. On the weak keys of Blowfish. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop*, number 1039 in Lecture Notes in Computer Science, pages 27–32, Cambridge, UK, February 21–23, 1996. Springer-Verlag. Also available as <http://www.dmi.ens.fr/~vaudenay/pub.html#Vau96a>.
- [28] Wietse Venema. TCP WRAPPER: Network monitoring, access control and booby traps. In *UNIX Security III Symposium*, pages 85–92, Baltimore, MD, September 14–16, 1992. USENIX Association. Also available as [ftp://ftp.porcupine.org/pub/security/tcp\\_wrapper.ps.Z](ftp://ftp.porcupine.org/pub/security/tcp_wrapper.ps.Z).
- [29] Machael J. Wiener. Efficient DES key search. In William Stallings, editor, *Practical Cryptography for Data Internetworks*, pages 31–79. IEEE Computer Society Press, 1996. Also available as <ftp://ripem.msu.edu/pub/crypt/docs/des-key-search.ps>.
- [30] Thomas Wu. The secure remote password protocol. In *1998 ISOC Network and Distributed System Security Symposium*, pages 97–111, March 1998. Also available as <ftp://srp.stanford.edu/pub/srp/srp.ps>.